CQF Exam Part One - June 2023 Cohort

In [1]:
```
#aggregating libary in one cell
import pandas as pd
import numpy as np
import scipy as sp
from scipy.stats import norm
from scipy.optimize import minimize
import matplotlib.pyplot as plt
import math

import cufflinks as cf
cf.set_config_file(offline=True, dimensions=(1000, 600), theme = 'hemanigans')

import plotly.express as px
px.defaults.width, px.defaults.height = 1000, 600
import plotly.graph_objects as go

import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.precision', 4)
```

important note:

- [x] Check in box represents the question has been answered (all questions should be answered)
- Some answer may add extra information than what is asked for purpose of self interest or checks
- Question 1 & 2 needs to use Analytical computation rather than numeral computation for optimization as it will result in non full mark. Numerical example approac in the 'Fundamentals of Optimization and Application to Portfolio Selection' by Dr Sebastien not allowed

Optimal Portfolio Allocation

An investment Universe of the following risky assets with a dependence structure (correlation) applies to all questions below as relevant:

Asset	μ	σ	w
A	0.05	0.07	w_1
B	0.07	0.28	w_2
C	0.15	0.25	w_3
D	0.22	0.31	w_4

$$\text{Corr} = \begin{pmatrix} 1.0 & 0.4 & 03 & 0.3 \\ 0.4 & 1.0 & 0.27 & 0.42 \\ 0.3 & 0.27 & 1.0 & 0.5 \\ 0.3 & 0.42 & 0.5 & 1.0 \end{pmatrix}$$

Question 1. Global Minimum Variance portfolio is obtained subject to the budget constraint:

$$\underset{w}{\text{argmin}} \frac{1}{2} w^{'} \Sigma w \ , \ \text{s.t.} \ w^{'} \hat{1} = 1$$

- [x] Derive the analytical solution for optimal allocations w^*. Provide full mathematical workings.
- [x] In the derivation, include the formula derivation for the Lagrangian multiplier.
- [x] Compute optimal allocations (global MV portfolio) for the given investment universe.

Budget constraints: $w^{'}\hat{1} = 1$ where, $\hat{1}$ is a 'n'th-element unit vector (vector of ones).

Lagrangian Formula will be as following:

$$L(w, \lambda) = \frac{1}{2} w^{'} \Sigma w + \lambda(w^{'}\hat{1} - 1)$$

The goal is to find the values of w and λ that minimize the Lagrangian formula $L(w, \lambda)$ above. To do this, we will need to take the partial derivatives of L with respect to w and λ to and find the optimal allocation w^*.

1st Order Condition:
Let us take the partial derivatives of $L(w, \lambda)$ with respect to w set them to 0

$$\frac{\partial L}{\partial w} = \Sigma w + \lambda \hat{1}$$

$$\frac{\partial L}{\partial w} | w = 0$$

$$\Sigma w + \lambda \hat{1} = 0$$

$$\Sigma w = -\lambda \hat{1}$$

2nd Order Condition:

$$\frac{\partial^2 L}{\partial w^2} = \frac{\partial L}{\partial w} (\frac{\partial L}{\partial w})$$

$$\frac{\partial^2 L}{\partial w^2} = \frac{\partial L}{\partial w} (\Sigma w + \lambda \hat{1})$$

$$\frac{\partial^2 L}{\partial w^2} = \Sigma$$

The second order condition, the Hessian of the objective function is the covariance matrix Σ, will be positive definite, which is what is needed for a minimimum.

Now, let us take the partial derivatives of $L(w, \lambda)$ with respect to λ and set them to 0

$$\frac{\partial L}{\partial \lambda} = w^{'}\hat{1} - 1$$

$$\frac{\partial L}{\partial \lambda} | \lambda = 0$$

$$w^{'}\hat{1} - 1 = 0$$

$$w^{'}\hat{1} = 1$$

When we combine these two equation below,

$$\begin{cases} \Sigma w = -\lambda \hat{1} \\ w^{'}\hat{1} = 1 \end{cases}$$

we get Lagrangian multipler λ as following:

$$\lambda = -1$$

With this, we can infer following formula on w^*:

$$\Sigma w^* = \lambda \hat{1}$$

Therefore, the solution of weight w^* that minimize the variance while satisfying the budget constraint above is:

$$w^* = \frac{\Sigma^{-1}\hat{1}}{\hat{1}^{'}\Sigma^{-1}\hat{1}}$$

where $\hat{1}^{'}$ is the transpose of unit vector $\hat{1}$

As per requested in the paper, we can get the optimal allocation (weights) using the analytical formulae instead of numerically identifying using the formula above to get weigh, return, and standard deviation. We will be using the code "from scipy.optimize import minimize" to compute optimal allocation (Global MV portfolio) for the given investment universe.

In [2]:
```
###Computing numerically based on input will result in deduction.
###Numerical example can be referenced in "Fundamentals of Optimization 2" by Dr Sebastien
###Reference and teaching on optimization in python is provided on "Python Labs - June 2023: 03

#create the matrix dataframe using pandas and numpy.
df_1 = pd.DataFrame(
    {'mu': pd.Series([0.05, 0.07, 0.15, 0.22], index = ['A', 'B', 'C', 'D']),
    'sigma': pd.Series([0.07, 0.28, 0.25, 0.31], index = ['A', 'B', 'C', 'D']),
     'w': pd.Series(['w1','w2','w3','w4'], index = ['A','B','C','D'])}
)

corr_matrix_1 = np.array([[1.0, 0.4, 0.3, 0.3],
                          [0.4, 1.0, 0.27, 0.42],
                          [0.3, 0.27, 1.0, 0.5],
                          [0.3, 0.42, 0.5, 1.0]])
```

In [3]:
```
#key informations
numofasset = df_1.shape[0]
mu = df_1['mu'].values
sigma = df_1['sigma'].values
w = df_1['w'].values

#generate covariance matrix from correlation matrix above
cov_matrix_1 = np.outer(sigma, sigma) * corr_matrix_1

#budget constraints only
constraints = ({'type': 'eq', 'fun': lambda w: np.sum(w) - 1},)

#portfolio weights bound allowing negative value and positive value
bounds = [(-4, 4)] * len(mu)

#objective of risk
def portfolio_risk(w, cov_matrix_1):
    return np.sqrt(np.dot(w.T, np.dot(cov_matrix_1, w)))
```

```
#adding initial weights
initial_weights = np.ones(numofasset) / numofasset

#solving optimization problem using the
result = minimize(portfolio_risk, initial_weights, args=(cov_matrix_1,), method='SLSQP', constra

optimal_weights = result.x
optimal_portfolio_risk = result.fun

print("Optimal Portfolio Weights:")
for asset, weight in zip(w, optimal_weights):
    print(f"{asset}: {weight:.4f}")

print("\nOptimal Portfolio Risk (σΠ):", optimal_portfolio_risk)
print("\nOptimal Portfolio Variance:", optimal_portfolio_risk**2)
```

```
Optimal Portfolio Weights:
w1: 1.0429
w2: -0.0421
w3: 0.0067
w4: -0.0075

Optimal Portfolio Risk (σΠ): 0.06905182945641972

Optimal Portfolio Variance: 0.004768155151278474
```

After running this code, you will get the optimized weights that correspond to the GMV portfolio. The GMV portfolio represents the allocation of assets that minimizes the portfolio variance, given the correlation structure and asset standard deviations provided in the data. This optimal portfolio variance gives you an indication of the lowest achievable risk for this set of assets, and would be located to the left hand side of the efficient frontier that has lowest risk (represented in x-axis).

Question 2. Consider the optimization for a target return m. There is no risk-free asset.

$$\underset{w}{\operatorname{argmin}} \frac{1}{2} w^{'} \Sigma w$$

$$w^{'} \hat{1} = 1$$

$$w^{'} \mu = m$$

- Compute w* and portfolio risk on $\sigma\Pi = \sqrt{w^{'} \Sigma w}$ for $m = 7\%$ for three levels of correlation.
- Correlation matrix X1, X1.3, X1.8, subject to individual correlation upper limit of 0.99, if the scaling results in correlation value above 1. Provide all results in a single table.

Budget constraints: $w^{'} \hat{1} = 1$ where, $\hat{1}$ is a 'n'th-element unit vector (vector of ones).
Return constraints: $w^{'} \mu = m$

Lagrange formula with Lagrange multiplier λ and γ will be as per below:

$$L(w, \lambda, \gamma) = \frac{1}{2} w^{'} \Sigma w + \lambda (m - w^{'} \mu) + \gamma (1 - w^{'} \hat{1})$$

1st Order Condition:

$$\frac{\partial L}{\partial w} = \Sigma w - \lambda \mu - \gamma \hat{1}$$

2nd Order Condition:

$$\frac{\partial^2 L}{\partial w^2} = \frac{\partial L}{\partial w} (\frac{\partial L}{\partial w})$$

$$\frac{\partial^2 L}{\partial w^2} = \frac{\partial L}{\partial w}(\Sigma w - \lambda\mu - \gamma\hat{1})$$

$$\frac{\partial^2 L}{\partial w^2} = \Sigma$$

The second order condition, the Hessian of the objective function is the covariance matrix Σ, which will be positive definite. Therefore we can conclude that we have reached the optimal weight vector w^*:

$$\frac{\partial L}{\partial w}|w^* = 0$$

$$\Sigma w^* - \lambda\mu - \gamma\hat{1} = 0$$

$$\Sigma w^* = \lambda\mu - \gamma\hat{1}$$

$$w^* = \Sigma^{-1}(\lambda\mu - \gamma\hat{1})$$

We can plot the above 1st Order condition to the constraints:

$$\mu^{'}\Sigma^{-1}(\lambda\mu + \gamma\hat{1}) = \lambda\mu^{'}\Sigma^{-1}\mu + \gamma\mu^{'}\Sigma^{-1}1 = m$$

$$\hat{1}^{'}\Sigma^{-1}(\lambda\mu + \gamma\hat{1}) = \lambda\hat{1}^{'}\Sigma^{-1}\mu + \gamma 1^{'}\Sigma^{-1}\hat{1} = 1$$

Because all are scalars, we can transpose each to A, B, C

$$\begin{cases} A = \hat{1}^{'}\Sigma^{-1}\hat{1} \\ B = \mu^{'}\Sigma^{-1}\hat{1} = \hat{1}^{'}\Sigma^{-1}\mu \\ C = \mu^{'}\Sigma^{-1}\mu \end{cases}$$

where $AC - B^2$ is not 0.

Lagrange multiplier becomes

$$\begin{cases} \lambda = \frac{Am - B}{AC - B^2} \\ \gamma = \frac{C - Bm}{AC - B^2} \end{cases}$$

adding back Lagrange multiplier λ and γ into the equation of w^* will give following:

$$w^* = \frac{1}{AC - B^2}\Sigma^{-1}[(A\mu - B\hat{1})m + (C\hat{1} - B\mu)]$$

Note that from the formula of 2nd order condition above, $w^* = \Sigma^{-1}(\lambda\mu - \gamma\hat{1})$, we are fiding the Global Minimum Variance portfolio (m) level so we can express this equation as:

$$w^*(m) = \Sigma^{-1}(\lambda(m)\mu - \gamma(m)\hat{1})$$

furthermore, since we have risk given as $\sigma\Pi = \sqrt{w^{'}\Sigma w}$, and the Lagrange multiplier solved as:

$$\begin{cases} \lambda = \frac{Am-B}{AC-B^2} \\ \gamma = \frac{C-Bm}{AC-B^2} \end{cases}$$

we can derive to formula:

$$\sigma_{\Pi}^2 = \frac{Am^2 - 2Bm + C}{AC - B^2}$$

Based on the above formula, let us compute $w*$ and portfolio risk on $\sigma\Pi = \sqrt{w' \Sigma w}$ for $m = 7\%$ for three levels of correlation matrix X1, X1.3, X1.8 of Corr above. The correlation is subject to individual correlation upper limit of 0.99, if the scaling universe results in correlation value above 1 (here we are keeping the initial correlation of 1 = 1).

In [4]:
```
#create the matrix dataframe using pandas and numpy similar to Question 1
df_2 = pd.DataFrame(
    {'mu': pd.Series([0.05, 0.07, 0.15, 0.22], index = ['A', 'B', 'C', 'D']),
    'sigma': pd.Series([0.07, 0.28, 0.25, 0.31], index = ['A', 'B', 'C', 'D']),
     'w': pd.Series(['w1','w2','w3','w4'], index = ['A','B','C','D'])}
)

corr_matrix_2 = np.array([[1.0, 0.4, 0.3, 0.3],
                          [0.4, 1.0, 0.27, 0.42],
                          [0.3, 0.27, 1.0, 0.5],
                          [0.3, 0.42, 0.5, 1.0]])
```

In [5]:
```
#scaling factor given from above as x1, x1.3, x1.8
scaling_factors = [1, 1.3, 1.8]

#create a box for scaled correlation matrix
scaled_corr_matrices = []

#use for loop to scale the correlation matrix with min to take anything below 0.99, and keep the
for scale in scaling_factors:
    scaled_corr_matrix = np.minimum(corr_matrix_2 * scale, 0.99)
    scaled_corr_matrix[np.where(corr_matrix_2 == 1)] = 1
    scaled_corr_matrices.append(scaled_corr_matrix)

#display the scaled correlation matrices
for idx, scaled_corr_matrix in enumerate(scaled_corr_matrices):
    print(f"Scaled Correlation Matrix (x{scaling_factors[idx]}):\n", scaled_corr_matrix)
    print("\n")
```

```
Scaled Correlation Matrix (x1):
 [[1.   0.4  0.3  0.3 ]
 [0.4  1.   0.27 0.42]
 [0.3  0.27 1.   0.5 ]
 [0.3  0.42 0.5  1.  ]]


Scaled Correlation Matrix (x1.3):
 [[1.    0.52  0.39  0.39 ]
 [0.52  1.    0.351 0.546]
 [0.39  0.351 1.    0.65 ]
 [0.39  0.546 0.65  1.   ]]


Scaled Correlation Matrix (x1.8):
 [[1.    0.72  0.54  0.54 ]
 [0.72  1.    0.486 0.756]
 [0.54  0.486 1.    0.9  ]
 [0.54  0.756 0.9   1.   ]]
```

In [6]:
```
###Computing numerically based on input will result in deduction so this method below using A, B
###Numerical example can be referenced in "Fundamentals of Optimization 2" and the excel file pr

#weight_2 = np.random.uniform(-1,1,len(df_2['mu']))
#weight_2 /= weight_2.sum()
#unit_ones = np.ones_like(weight_2)
```

```
#m = 0.07

#cov_matrix_2 = np.outer(sigma, sigma) * corr_matrix_2
#inv_cov_matrix_2 = np.linalg.inv(cov_matrix_2)

#A = unit_ones.T @ inv_cov_matrix_2 @ unit_ones
#B = df_2['mu'].T @ inv_cov_matrix_2 @ unit_ones
#C = df_2['mu'].T @ inv_cov_matrix_2 @ df_2['mu']

#table_data_2 = []

#wt = inv_cov_matrix_2 @ (df_2['mu']-np.dot(m,unit_ones)/(B-A*m))
#sigmat = np.sqrt((A*m**2-2*B*m+C)/(A*C-B**2))

#table_data_2.append({'Return':m, 'Weights allocation':np.round(wt,3), 'σΠ':sigmat})
#result_table = pd.DataFrame(table_data_2)
#print(result_table.to_string(index=False))
```

As per requested in the paper, we can get the optimal allocation (weights) using the analytical formulae below instead of numerically identifying A, B, C and using the formula above to get weigh, return, and standard deviation. We will be using the code "from scipy.optimize import minimize" to compute optimal allocation.

In [7]:

```
#I am using the scipy.optimize specifically minimize() function to conduct analytical portfolio
#Reference and teaching is provided on "Python Labs - June 2023: 03 Portfolio Optimization - 2"

#data similar to above Q1
df_2 = pd.DataFrame(
    {'mu': pd.Series([0.05, 0.07, 0.15, 0.22], index=['A', 'B', 'C', 'D']),
     'sigma': pd.Series([0.07, 0.28, 0.25, 0.31], index=['A', 'B', 'C', 'D']),
     'w': pd.Series(['w1', 'w2', 'w3', 'w4'], index=['A', 'B', 'C', 'D'])}
)

#compute scaling factors
scaling_factors = [1, 1.3, 1.8]

#target return of m=7%
m = 0.07

#create a shell for results_2 for adding 3 different results
results_2 = []

#use for loop to find all correlation matrices and find its
for scale in scaling_factors:
    #correlation matrix similar to above
    corr_matrix_2 = np.array([[1.0, 0.4, 0.3, 0.3],
                              [0.4, 1.0, 0.27, 0.42],
                              [0.3, 0.27, 1.0, 0.5],
                              [0.3, 0.42, 0.5, 1.0]])

    #scaling and adding rule of cap at 0.99, and keep the value 1 if initial value was 1 (so tha
    scaled_corr_matrix = np.minimum(corr_matrix_2 * scale, 0.99)
    scaled_corr_matrix[np.where(corr_matrix_2 == 1)] = 1

    #use each std deviation and each scaled correlation matrix to compute compute covariance mat
    cov_matrix_2 = np.outer(df_2['sigma'], df_2['sigma']) * scaled_corr_matrix

    #objective function to minimize which is dependent on w
    def objective_function(w):
        portfolio_std = np.sqrt(w @ cov_matrix_2 @ w)
        return portfolio_std

    #budget constraints and return constraints added as constraint to be used later in minimize(
    constraints = ({'type': 'eq', 'fun': lambda w: np.sum(w) - 1},
                   {'type': 'eq', 'fun': lambda w: np.sum(w * df_1['mu']) - m})

    #optimization with initial random weights and can be negative
    initial_weights = np.random.uniform(-1, 1, len(df_1))
    result = minimize(objective_function, initial_weights,
                      method='SLSQP', constraints=constraints)

    #if we see result, x represents the optimized weight. we will name result.x accordingly
    optimized_weights = result.x

    #create a dictionary with the asset weights so I can use this later in the table result
    asset_weights = {asset: weight for asset, weight in zip(df_1['w'], optimized_weights)}
```

```
    #get portfolio standartd deviation
    optimized_portfolio_std = np.sqrt(optimized_weights @ cov_matrix_2 @ optimized_weights)

    results_2.append({
        'Scaling Factor': scale,
        **asset_weights,
        'Optimal Portfolio Risk (σΠ)': optimized_portfolio_std
    })

#create the dataframe and display the dataframe
results_df = pd.DataFrame(results_2)
print(results_df.to_string(index=False) )
```

```
 Scaling Factor     w1      w2      w3     w4  Optimal Portfolio Risk (σΠ)
            1.0 0.9242 -0.0730  0.0547 0.0940                       0.0774
            1.3 0.9965 -0.1352  0.0124 0.1263                       0.0765
            1.8 1.4509 -0.4077 -0.5071 0.4639                       0.0412
```

Question 3. "Evaluating the P&L more frequently make it appear more risky than it actually is." make the following simple computations to demonstrate this statement.

- [x] Write down the formula for Sharpe Ratio and identify main parameter scaled with time.
- [x] Compute Daily, Monthly, and Quarterly Sharpe Ratio for Annualised SR of 0.54. No other inputs.
- [x] Convert each Sharpe Ratio into Loss Probability (daily, monthly, quarterly, anual), using

$$\Pr(\text{P\&L} < 0) = \Pr(x < \text{-SR})$$

where, x is the standard Normal random variable.

The Sharpe Ratio is measuring the risk-adjusted return of an investment or a portfolio. It is computed with following formula:

$$\text{Sharpe Ratio}_p = \frac{\mu_p - r_f}{\sigma_p}$$

Where,

- μ_p = Expected Return of Portfolio
- r_f = Risk-free Rate
- σ_p = Standard Deviation of Portfolio Returns

The main parameter scaled with time is the μ which is the Expected return of portfolio or an investment.

In [8]:
```
#computing the SR at different period using the annualized SR of 0.54.
sr_annual = 0.54
sr_daily = sr_annual/math.sqrt(252) #1 year with assumption of  252 business day
sr_monthly = sr_annual/math.sqrt(12) #1 year with 12 month
sr_quarterly = sr_annual/math.sqrt(4) #1 year with 4 quarter

#print values
print("Sharpe Ratios:")
print("Daily:", sr_daily)
print("Monthly:", sr_monthly)
print("Quarterly:", sr_quarterly)
print("Annual:", sr_annual)
```

```
Sharpe Ratios:
Daily: 0.03401680257083045
Monthly: 0.15588457268119899
Quarterly: 0.27
Annual: 0.54
```

In [9]:
```
#using formula above to get the z-scores for each sharpe ratio
z_daily = -sr_daily
z_monthly = -sr_monthly
z_quarterly = -sr_quarterly
z_annual = -sr_annual

#calculating probability using the norm from scipy.stats
loss_prob_daily = norm.cdf(z_daily)
loss_prob_monthly = norm.cdf(z_monthly)
```

```
loss_prob_quarterly = norm.cdf(z_quarterly)
loss_prob_annual = norm.cdf(z_annual)

#print values
print("Loss Probability:")
print("Probability of daily loss:", loss_prob_daily*100,"%")
print("Probability of monthly loss:", loss_prob_monthly*100,"%")
print("Probability of quarterly loss:", loss_prob_quarterly*100,"%")
print("Probability of annual loss:", loss_prob_annual*100,"%")
```

```
Loss Probability:
Probability of daily loss: 48.64318759705117 %
Probability of monthly loss: 43.80620029308649 %
Probability of quarterly loss: 39.35801268019605 %
Probability of annual loss: 29.4598516215698 %
```

The statement "Evaluating the P&L more frequently make it appear more risky than it actually is." refers to fact that higher frequency observation can magnify the volatility and fluctuations of returns. As per data above, both through Sharpe Ratio and Loss Probability, we can see that more frequent observation such as daily frequency, yields a lower SR which implies higher perceived risk, and higher loss probability representing the that it has more likelyhood of experiencing a loss. We are deriving this statement based on calculated SR, with the assumption of returns being normally distributed.

Question 4. Instead of computing the optimal allocations analytically, let's conduct an experiment. Generate above 700 random allocation sets: 4 X 1 vectors. Those will not be optimal and can be negative.

- [x] Standardise each set to satisfy $w^{'}1 = 1$. In fact, generate 3 allocations and compute the 4th.
- [x] For each set, compute $\mu\Pi = W^{'}\mu$ and $\sigma\Pi = \sqrt{w'\Sigma w}$.
- [x] Plot the cloud of points, $\mu\Pi$ vertically on $\sigma\Pi$ horizontally. Explain this plot

In [10]:
```
#Using the investment universe at "Optimal Portfolio Allocation" section above, and naming it di
df_4 = pd.DataFrame(
    {'mu': pd.Series([0.05, 0.07, 0.15, 0.22], index = ['A', 'B', 'C', 'D']),
    'sigma': pd.Series([0.07, 0.28, 0.25, 0.31], index = ['A', 'B', 'C', 'D']),
     'omega': pd.Series(['w1','w2','w3','w4'], index = ['A','B','C','D'])}
)

corr_matrix_4 = np.array([[1.0, 0.4, 0.3, 0.3],
                          [0.4, 1.0, 0.27, 0.42],
                          [0.3, 0.27, 1.0, 0.5],
                          [0.3, 0.42, 0.5, 1.0]])
```

In [11]:
```
#number of experiment simulations
simulations = 700

#intialize the list
numofasset = len(df_4)
μΠ = [] ; σΠ=[]

#siumlate the experiment with 700 random allocation
for i in range(simulations):

    #generate random weights for each random number of asset
    weights = np.random.random(numofasset)

    #set weight such athat sum of weight = 1
    weights /= weights.sum()

    #simulation stats
    μΠ.append(weights.T @ df_4['mu'])
    sigma_4 = np.diag(np.array(df_4['sigma'])@ np.eye(numofasset))
    covariance_4 = sigma_4 @ corr_matrix_4 @ sigma_4
    σΠ.append(np.sqrt(weights.T @ covariance_4 @ weights))

#plot the simulation in dataframe format and include sharpe ratio for further explanation
portfolio_data = pd.DataFrame({'μΠ':np.array(μΠ), 'σΠ':np.array(σΠ)})
portfolio_data['SR'] = portfolio_data['μΠ']/portfolio_data['σΠ']
```

```
#find the min variance valuein the table
portfolio_data.iloc[portfolio_data.σΠ.idxmin()]

#show all 700 simulation in table
portfolio_data
```

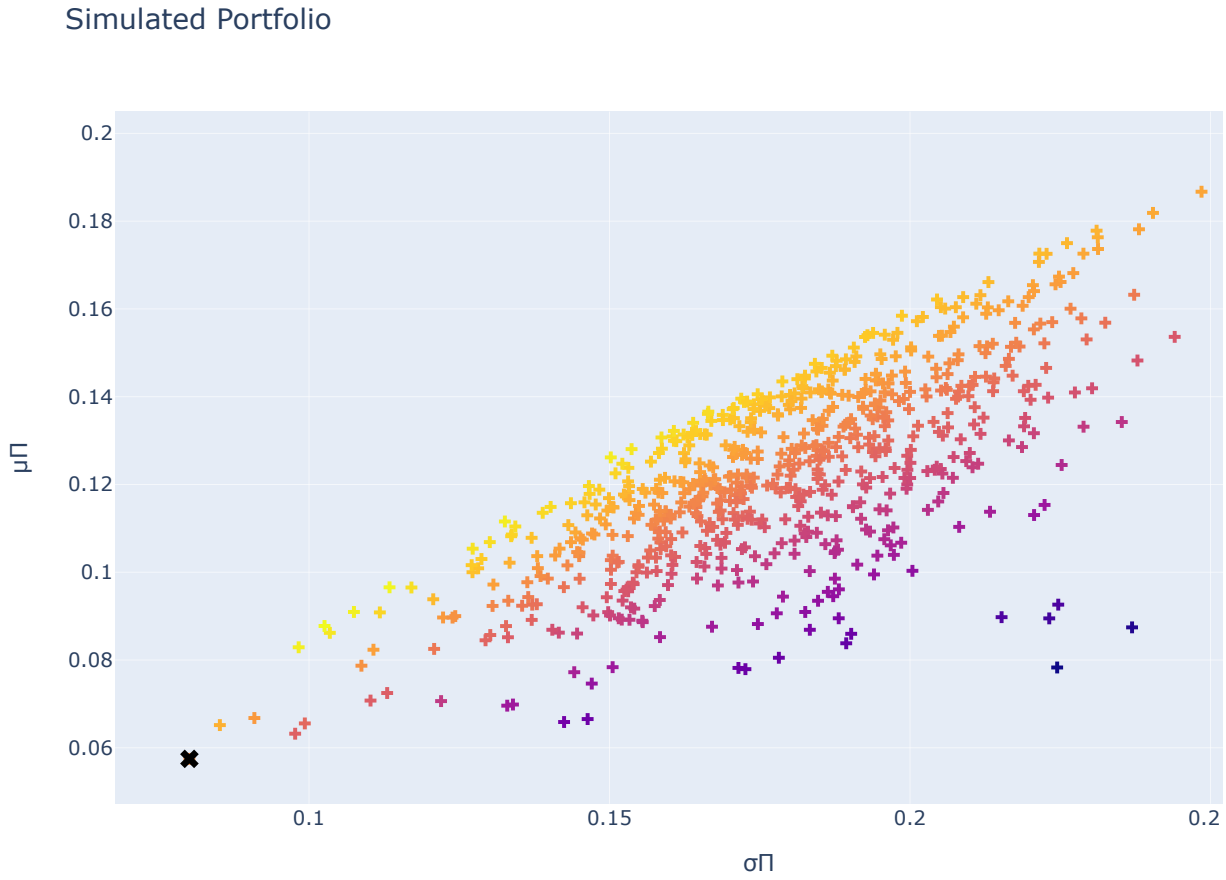
Out[11]:

	μΠ	σΠ	SR
0	0.1333	0.2190	0.6085
1	0.1181	0.1560	0.7573
2	0.1381	0.1942	0.7111
3	0.1190	0.1994	0.5968
4	0.1138	0.1594	0.7142
...	...	...	...
695	0.1090	0.1694	0.6430
696	0.1042	0.1808	0.5764
697	0.1196	0.1533	0.7806
698	0.0991	0.1385	0.7156
699	0.1188	0.1855	0.6405

700 rows × 3 columns

In [12]:

```
#plot simulated portfolio
fig_4 = px.scatter(
    portfolio_data, x='σΠ', y='μΠ', color = 'SR',
    labels={'σΠ':'σΠ', 'μΠ':'μΠ', 'SR':'Sharpe Ratio'},
    title="Simulated Portfolio"
).update_traces(mode='markers', marker=dict(symbol='cross'))

#if we want to find the minimum risk, we can mark the most smallest simulation as per below
fig_4.add_scatter(
    mode='markers',
    x=[portfolio_data.iloc[portfolio_data.σΠ.idxmin()]['σΠ']],
    y=[portfolio_data.iloc[portfolio_data.σΠ.idxmin()]['μΠ']],
    marker=dict(color='black', size=10, symbol='x'),
    name='Minimum σΠ'
).update(layout_showlegend=False)

#show spikes
fig_4.update_xaxes(showspikes=True)
fig_4.update_yaxes(showspikes=True)

fig_4.show()
```



The above simulated plot represents plot of retun and risk (which we represent using 'σ') of 700 different portfolio based on randomly generated risky asset weight allocation. The plot marked as 'x' in black represents value at which we have the lowest risk, and are mnamed as point of 'Minimum $\sigma\Pi$'. Furthermore, if we take the highest return per given standard deviation we should be able to draw the efficient frontier of the portfolio of these risky asset.

Question 5. NASDAQ100 data privided (2012-2023) for you to implmenet the backtesting of 99%/10day Value at Risk and report the following:

- [x] Compute the rolling standard deiation σt from 21 daily returns.
- [x] Timescale of that σt remains 'daily' regardless of how many returns are in the sample To make projection, use the additivity of variance $\sigma_{10D} = \sqrt{\sigma_t^2 * 10}$
- [x] A breach occurs when the forward realized 10-day return is below the VaR_t quantity.

$$r_{10D,t+10} < VaR_{10D,t}$$

means breach, given both number are negative.

VaR is fixed at time t and compared to the return realised from t to t+10, computed $ln(S_{t+10}/S_t)$. Alternatively, you can compare to ln(S{t+11}/S{t+1}) but state this assumption in your report upfront.

In [13]:
```
#import the csv file called 'nasdaq100.csv'
nasdaq = pd.read_csv('nasdaq100.csv', index_col='Date', parse_dates=True, infer_datetime_format=
```

In [14]:
```
nasdaq.head()
```

Out[14]:

Date	Closing Price
2017-12-01	6337.8701
2017-12-04	6263.7002
2017-12-05	6265.1099
2017-12-06	6293.0498
2017-12-07	6316.2798

(a) The count of count and percentage of VaR breaches.

In [15]:
```
#compute the logreturn for 1d, 10d, stdv, and var (with c = 0.99, and t=10).
#code below will be using 10D return realized using the ln(St+11/St+1) for the VaR fixed at time
nasdaq_1d_return = np.log(nasdaq/nasdaq.shift(1))
rolling_stdv = nasdaq_1d_return.rolling(window=21).std()
nasdaq_10d_return = np.log(nasdaq.shift(-11)/nasdaq.shift(-1))
c = 0.99 #confidence interval
t = 10 #day
var = norm.ppf(1-c)* rolling_stdv * np.sqrt(t)

#combine return vs VaR in df to count breaches and percentage of breaches.
backtest = pd.merge(nasdaq_10d_return, var , on='Date').dropna()
backtest = backtest.rename(columns={'Closing Price_x':'Return','Closing Price_y':'VaR'})

return_var_breach = (backtest['Return']<backtest['VaR'])
var_breach_count = return_var_breach.sum()
var_breach_percent = return_var_breach.mean() #or var_breach_count/len(return_breach_count)

#printing the result of the backtest and expected number
print(f"Expected Breach Count: {round(len(nasdaq_1d_return['Closing Price'])*(1-c), 2)}")
print(f"Expected Breach Percentage: {round((1-c), 5)*100}%")
print(f"VaR Total Breach Count: {var_breach_count}")
print(f"VaR Total Breach Percentage: {round(var_breach_percent, 5)*100}%")
```

```
Expected Breach Count: 13.8
Expected Breach Percentage: 1.0%
VaR Total Breach Count: 41
VaR Total Breach Percentage: 3.042%
```

The code above is simply manually counting how many breaches there were comparing the prediction of 99%/10D VaR vs the 10D forward returns.

(b) The count of consecutive VaR breaches. (1, 1, 1 indicates two consecutive occurences)

In [16]:
```
#using for loops to keep track of the number variable for consecutive breaches.
#here we are leveraging the breach ('True') and no breach ('False')
consecutive_breaches = 0
consecutive_counts = []

for breach in return_var_breach:
    if breach:
        consecutive_breaches += 1
    else:
        if consecutive_breaches > 0:
            consecutive_counts.append(consecutive_breaches)
            consecutive_breaches = 0

if consecutive_breaches > 0:
    consecutive_counts.append(consecutive_breaches)

#store to dictionary called 'consecutive_breach_results'
consecutive_breach_results = []

for count in consecutive_counts:
    consecutive_breach_results.append({'Number of consecutive breach': count, 'Count of breach h

#add dataframe and sort,filter accordingly to showcase in tabular format
consecutive_breach_results_df = pd.DataFrame(consecutive_breach_results)
```

```
consecutive_breach_results_df = consecutive_breach_results_df.drop_duplicates().sort_values(by='
print(consecutive_breach_results_df.to_string(index=False))
```

```
 Number of consecutive breach  Count of breach happening
                            1                          9
                            2                          6
                            3                          1
                            4                          1
                            5                          1
                            8                          1
```

(C) Provide a plot which: identifies the breaches visually (crosses or other marks) and properly label axis X with at least years.

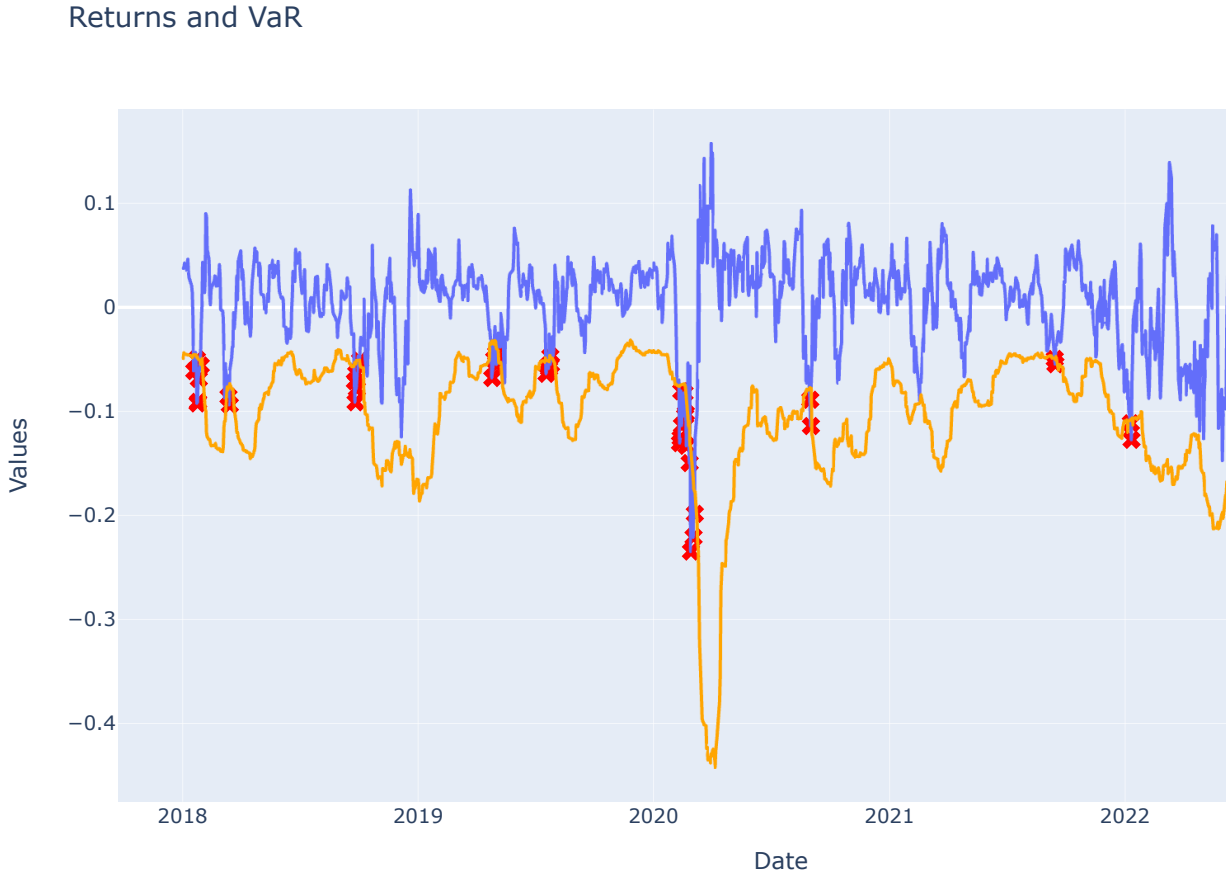
In [17]:
```
#adding the 'Breach' column, representing bools of 'True' & 'False'
backtest['Breach'] = return_var_breach
backtest
```

Out[17]:

Date	Return	VaR	Breach
2018-01-03	0.0372	-0.0495	False
2018-01-04	0.0373	-0.0439	False
2018-01-05	0.0421	-0.0456	False
2018-01-08	0.0355	-0.0455	False
2018-01-09	0.0374	-0.0457	False
...	...	...	...
2023-05-05	0.0411	-0.0806	False
2023-05-08	0.0351	-0.0800	False
2023-05-09	0.0190	-0.0809	False
2023-05-10	0.0402	-0.0816	False
2023-05-11	0.0694	-0.0797	False

1348 rows × 3 columns

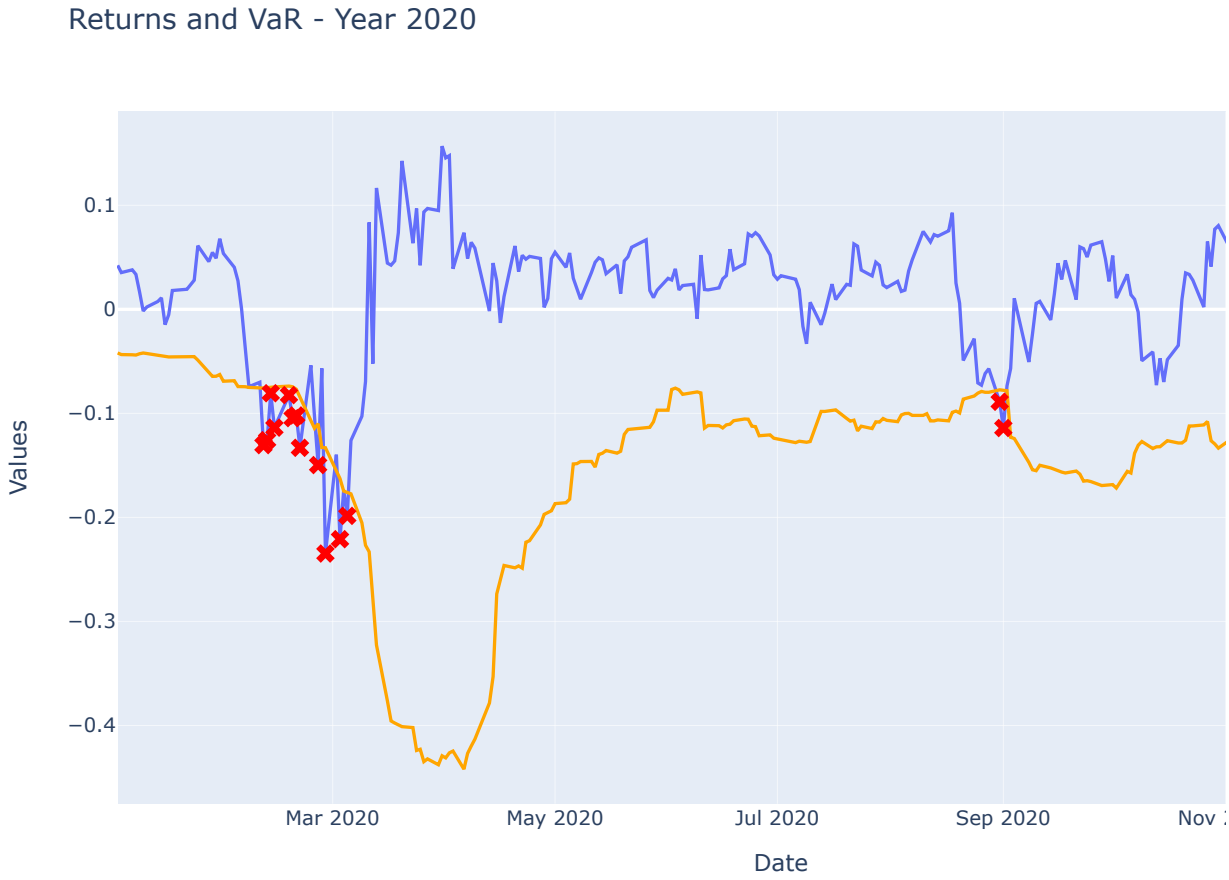
In [18]:
```
#create a scatter plot using plotly
fig_5_c = px.line(backtest, x=backtest.index, y=['Return', 'VaR'], title='Returns and VaR')

#mark breaches with red 'x'
breach_indices = backtest[return_var_breach].index
fig_5_c.add_trace(go.Scatter(x=breach_indices, y=backtest.loc[breach_indices, 'Return'],
                         mode='markers', marker=dict(symbol='x', size=10, color='red'),
                         name='Breach'))

#update x-axis and y-axis labels to show more clear information
fig_5_c.update_xaxes(title='Date', showspikes=True)
fig_5_c.update_yaxes(title='Values', showspikes=True)

#change the color of the VaR line, as we have red 'x' for the Braech
fig_5_c.update_traces(selector=dict(name='VaR'), line=dict(color='orange'))

#show chart of Return, VaR and mark 'x' on Breaches
fig_5_c.show()
```



(d) In your own words, describe the sequence of breaches caused by COVID pandemic news in 2020-Feb versus 2020-Mar

In [19]:

```
#filter data for the year 2020. or using plotly you can highlight in chart 5(c) directly.
filtered_data = backtest['2020-01':'2020-12']

#create a line plot for Return and VaR
fig_5_d = px.line(filtered_data, x=filtered_data.index, y=['Return', 'VaR'], title='Returns and

#mark breaches with red 'x'
breach_indices = filtered_data[return_var_breach].index
fig_5_d.add_trace(go.Scatter(x=breach_indices, y=filtered_data.loc[breach_indices, 'Return'],
                         mode='markers', marker=dict(symbol='x', size=10, color='red'),
                         name='Breach'))

#update x-axis and y-axis labels to show more clear information
fig_5_d.update_xaxes(title='Date', showspikes=True)
fig_5_d.update_yaxes(title='Values', showspikes=True)

#change the color of the VaR line, as we have red 'x' for the Braech
fig_5_d.update_traces(selector=dict(name='VaR'), line=dict(color='orange'))

#show chart of Return, VaR and mark 'x' on Breaches
fig_5_d.show()
```



The diagram above shows the two financial metrics which is "Return" and the "Value at Risk (VaR)" over time. Return (represented in blue) shows the historical percentage of profit or loss generated on the Nasdaq100 Index over time, while VaR (represented in orange) shows the estimated maximum potential loss of Nasdaq100 can possibly give at 99% confidence interval. Note that this VaR value is computed based on volatility and historical return distribution and refers to the estimate the worst-case scenario under normal market conditions.

"Returns and VaR - Year 2020" diagram shows multiple breaches concentrated on the month of February which falls when COVID pandemic news caused rapid decline in the financial market. Other indicator of which characterized this event was the markets high level of volatility, decline in overall prices in asset, and also uncertanty of the investment world.

Based on the market condition, we can deduct that investment return for Nasdaq100 constituents experienced losses, and had high level of volatility, causing the return to surpass the maximum level of loss that can occur based on historical and statistical data. It has been showing periods of uncertainty, affecting investor behavior to push panic selling and mass liquidation of asset. From March onwards, we see an increase in return, which shows the recovery in the market that might have been driven by factors like government interventions, financial stimulus or could possibly be the investor confidence coming back as there are more information about the pandemic available.

These event like COVID pandemic news can be considered as an event that is unexpected or very rare which would cause massive impact to market. This also can be called 'Black Swan" event. Black swan events also usually lead to 'Tail Risk' where event falls outside of the normal distribution of the return that models estimation. Therefore, these breach underscores the limitation of traditional risk modeling using historical and statistical data, and shows complexities of managing risk at time of extreme events.

Question 6. Re-implement bactesting using the method above, recompute $\text{VaR}_{10D,t}$ but, with the input of EWMA σ^2_{t+1}. Use the variance for the entire dataset to initialise the scheme

$$\sigma^2_{t+1|t} = \lambda\sigma^2_{t|t-1} + (1-\lambda)r^2_t$$

with, $\lambda = 0.72$ value set to minimise out of sample forcasting error.

Hint: computation of EWMA σ^2_{t+1} is not sufficient, proceed to compute $\text{VaR}_{10D,t}$ and count breaches in VaR

In [20]:
```
#lets create a table similar to the course by Dr.Richard Diamond - Tutorial:"03 Statistical Esse
lambda_value = 0.72
horizon_t_trading_day = 10
df_6 = nasdaq_1d_return.rename(columns={'Closing Price':'Log-Return'}).dropna()
df_6['Squared Return'] = df_6**2

#get the average variance by averaging the squared return, and input the avg variance for start
avg_variance = np.average(df_6['Squared Return'])
df_6['Variance Estimate'] = np.nan
df_6.loc['2017-12-04', 'Variance Estimate'] = avg_variance

#reference the average variance and get var_estimate(t) = var_estimate(t-1)*lambda_value+squared
#use the for loop to impute all the Variance Estimate value based on above formula
for i in range(1, len(df_6)):
                df_6.loc[df_6.index[i], 'Variance Estimate'] = df_6.loc[df_6.index[i-1], 'Variar

#get the stdev and scale the stdev to time horizon as as variance we computed above is based of
df_6['Estimate σ, 1D'] = np.sqrt(df_6['Variance Estimate'])
df_6['Estimate T-days ahead']= df_6['Estimate σ, 1D'] * np.sqrt(horizon_t_trading_day)

#show the table
df_6
```

Out[20]:

Date	Log-Return	Squared Return	Variance Estimate	Estimate σ, 1D	Estimate T-days ahead
2017-12-04	-0.0118	1.3857e-04	2.6938e-04	0.0164	0.0519
2017-12-05	0.0002	5.0638e-08	2.3276e-04	0.0153	0.0482
2017-12-06	0.0044	1.9800e-05	1.6760e-04	0.0129	0.0409
2017-12-07	0.0037	1.3576e-05	1.2621e-04	0.0112	0.0355
2017-12-08	0.0045	1.9971e-05	9.4676e-05	0.0097	0.0308
...	...	...	...	...	...
2023-05-22	0.0033	1.1189e-05	1.0196e-04	0.0101	0.0319
2023-05-23	-0.0129	1.6582e-04	7.6545e-05	0.0087	0.0277
2023-05-24	-0.0050	2.4903e-05	1.0154e-04	0.0101	0.0319
2023-05-25	0.0243	5.8844e-04	8.0083e-05	0.0089	0.0283
2023-05-26	0.0255	6.4981e-04	2.2242e-04	0.0149	0.0472

1379 rows × 5 columns

(a) The count of count and percentage of VaR breaches.

In [21]:
```
#similar to method in Q5_(a), we can do exactly similar method to compute the breach
nasdaq_10d_return_6_a = np.log(nasdaq.shift(-11)/nasdaq.shift(-1))
c_6 = 0.99 #confidence interval
t_6 = 10 #day
var_6_a = norm.ppf(1-c)* df_6['Estimate T-days ahead'] #we have computed 10D scaled estimate va


#combine return vs VaR in df to count breaches and percentage of breaches.
backtest_6_a = pd.merge(nasdaq_10d_return_6_a, var_6_a , on='Date').dropna()
backtest_6_a
backtest_6_a = backtest_6_a.rename(columns={'Closing Price':'Return','Estimate T-days ahead':'10

return_var_breach_6_a = (backtest_6_a['Return'] < backtest_6_a['10D Estimated VaR'])
var_breach_count_6_a = return_var_breach_6_a.sum()
var_breach_percent_6_a = return_var_breach_6_a.mean()

#printing the result of the backtest and expected number
```

```
print(f"Expected Breach Count: {round(len(df_6.index)*(1-c), 2)}")
print(f"Expected Breach Percentage: {round((1-c), 4)*100}%")
print(f"VaR Total Breach Count: {var_breach_count_6_a}")
print(f"VaR Total Breach Percentage: {round(var_breach_percent_6_a, 4)*100}%")
```

```
Expected Breach Count: 13.79
Expected Breach Percentage: 1.0%
VaR Total Breach Count: 54
VaR Total Breach Percentage: 3.95%
```

(b) The count of consecutive VaR breaches. (1, 1, 1 indicates two consecutive occurences)

In [22]:
```
#using for loops to keep track of the number variable for consecutive breaches.
#here we are leveraging the breach ('True') and no breach ('False')
consecutive_breaches_6_b = 0
consecutive_counts_6_b = []

for breach in return_var_breach_6_a:
    if breach:
        consecutive_breaches_6_b += 1
    else:
        if consecutive_breaches_6_b > 0:
            consecutive_counts_6_b.append(consecutive_breaches_6_b)
            consecutive_breaches_6_b = 0

if consecutive_breaches_6_b > 0:
    consecutive_counts_6_b.append(consecutive_breaches_6_b)

#store to dictionary called 'consecutive_breach_results_6_b' similar to way we did for 5_b
consecutive_breach_results_6_b = []

for count in consecutive_counts_6_b:
    consecutive_breach_results_6_b.append({'Number of consecutive breach': count, 'Count of brea

#add dataframe and sort,filter accordingly to showcase in tabular format
consecutive_breach_results_df_6_b = pd.DataFrame(consecutive_breach_results_6_b)
consecutive_breach_results_df_6_b = consecutive_breach_results_df_6_b.drop_duplicates().sort_val
print(consecutive_breach_results_df_6_b.to_string(index=False))
```

```
 Number of consecutive breach  Count of breach happening
                            1                          8
                            2                          7
                            3                          4
                            4                          1
                            8                          2
```

(C) Provide a plot which: identifies the breaches visually (crosses or other marks) and properly label axis X with at least years.

In [23]:
```
#adding the 'Breach' column, representing bools of 'True' & 'False' to draw in plotly
backtest_6_a['Breach'] = return_var_breach_6_a
backtest_6_a
```

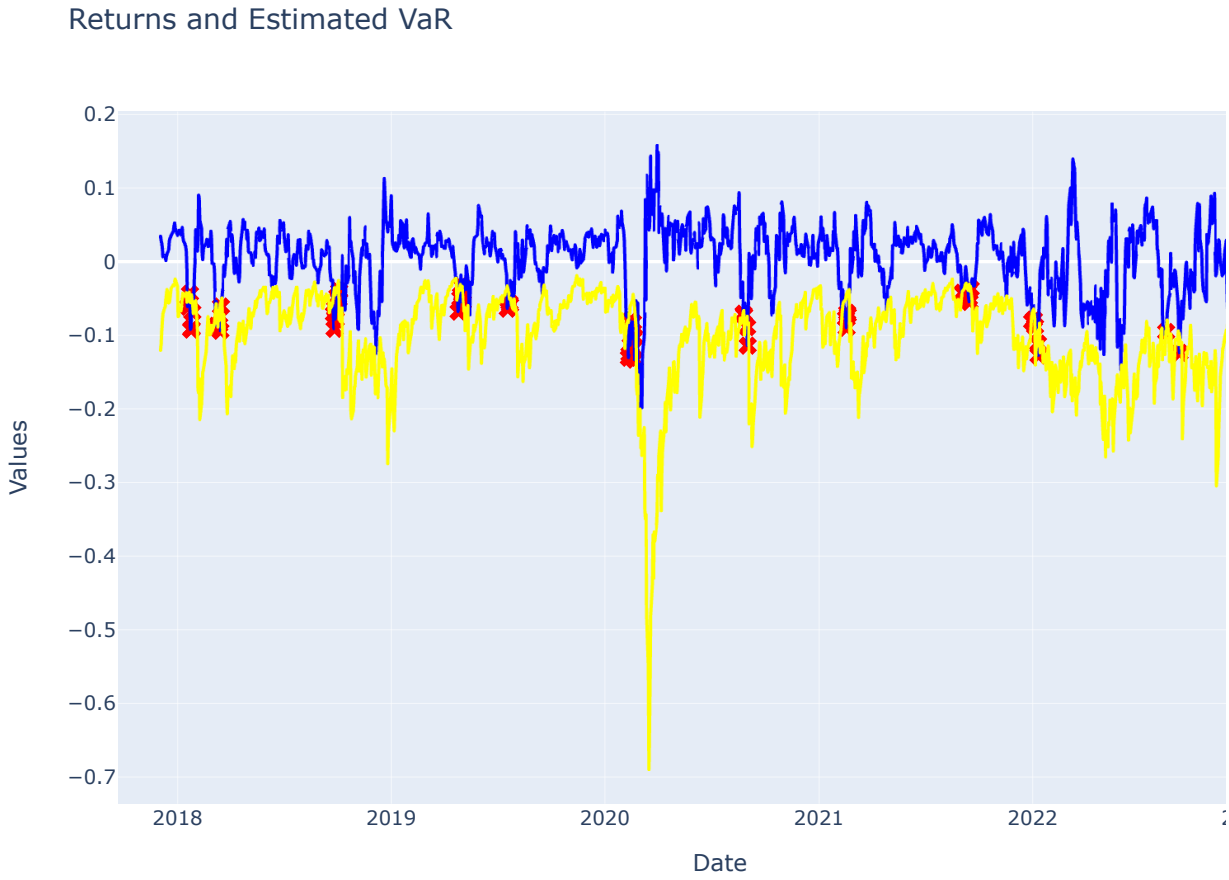
Out[23]:

Date	Return	10D Estimated VaR	Breach
2017-12-04	0.0338	-0.1207	False
2017-12-05	0.0281	-0.1122	False
2017-12-06	0.0245	-0.0952	False
2017-12-07	0.0188	-0.0826	False
2017-12-08	0.0061	-0.0716	False
...	...	...	...
2023-05-05	0.0411	-0.0661	False
2023-05-08	0.0351	-0.0994	False
2023-05-09	0.0190	-0.0849	False
2023-05-10	0.0402	-0.0768	False
2023-05-11	0.0694	-0.0781	False

1368 rows × 3 columns

In [24]:

```
#create a scatter plot using plotly
fig_6_c = px.line(backtest_6_a, x=backtest_6_a.index, y=['Return', '10D Estimated VaR'], title='

#mark breaches with red 'x'
breach_indices_6_c = backtest_6_a[return_var_breach_6_a].index
fig_6_c.add_trace(go.Scatter(x=breach_indices_6_c, y=backtest_6_a.loc[breach_indices_6_c, 'Retur
                         mode='markers', marker=dict(symbol='x', size=10, color='red'),
                         name='Breach'))

#update x-axis and y-axis labels to show more clear information
fig_6_c.update_xaxes(title='Date', showspikes=True)
fig_6_c.update_yaxes(title='Values', showspikes=True)

#change the color of the VaR line, as we have red 'x' for the Braech
fig_6_c.update_traces(selector=dict(name='10D Estimated VaR'), line=dict(color='yellow'))
fig_6_c.update_traces(selector=dict(name='Return'), line=dict(color='blue'))

#show chart of Return, VaR and mark 'x' on Breaches
fig_6_c.show()
```



(d) Briefly (3-4 lines), discuss the impact of λ on smoothness of EWMA-predicted volatility

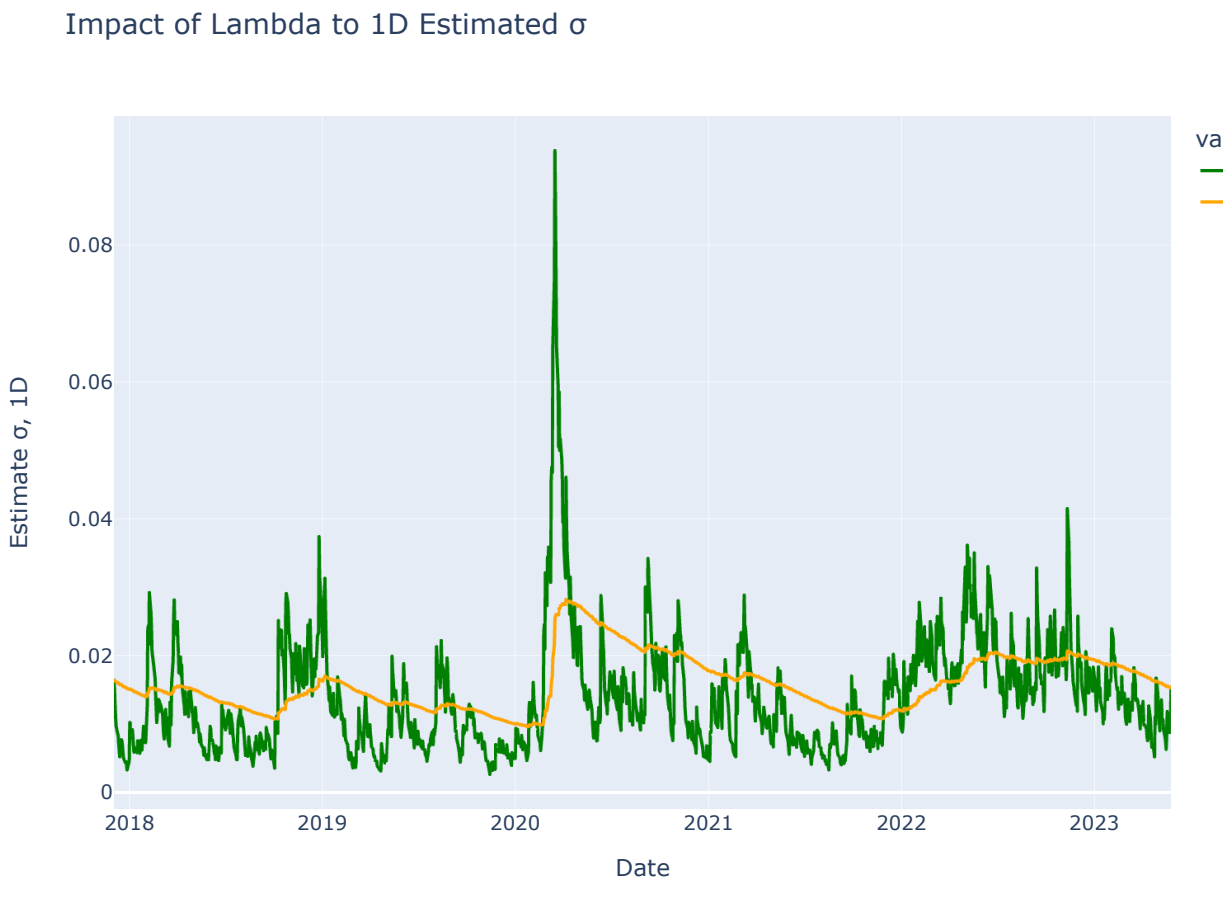
In [25]:
```
#to show the difference between the impact of EWMA-predicted volatility, we can add a new colum
new_lambda_value = 0.99
df_6['Variance Estimate with Lambda=0.99'] = np.nan
df_6.loc['2017-12-04', 'Variance Estimate with Lambda=0.99'] = avg_variance

#reference the average variance and get var_estimate(t) = var_estimate(t-1)*lambda_value+squared
#use the for loop to impute all the Variance Estimate value based on above formula
for i in range(1, len(df_6)):
                df_6.loc[df_6.index[i], 'Variance Estimate with Lambda=0.99'] = df_6.loc[df_6.in

df_6['Estimate σ with Lambda=0.99, 1D'] = np.sqrt(df_6['Variance Estimate with Lambda=0.99'])
columns_6_d = ['Estimate σ, 1D', 'Estimate σ with Lambda=0.99, 1D']
table_6_d = df_6[columns_6_d]
table_6_d = table_6_d.rename(columns={'Estimate σ, 1D':'Estimate σ with Lambda=0.72, 1D'})

#pull the two data only. Note that we are not pulling VaR values, but only estimated σ value der
table_6_d
```

Out[25]:

Date	Estimate σ with Lambda=0.72, 1D	Estimate σ with Lambda=0.99, 1D
2017-12-04	0.0164	0.0164
2017-12-05	0.0153	0.0164
2017-12-06	0.0129	0.0163
2017-12-07	0.0112	0.0162
2017-12-08	0.0097	0.0161
...	...	...
2023-05-22	0.0101	0.0154
2023-05-23	0.0087	0.0154
2023-05-24	0.0101	0.0153
2023-05-25	0.0089	0.0153
2023-05-26	0.0149	0.0154

1379 rows × 2 columns

In [26]:
```
#create a scatter plot using plotly
fig_6_d = px.line(table_6_d, x=table_6_d.index, y=['Estimate σ with Lambda=0.72, 1D', 'Estimate

#update x-axis and y-axis labels to show more clear information
fig_6_d.update_xaxes(title='Date', showspikes=True)
fig_6_d.update_yaxes(title='Estimate σ, 1D', showspikes=True)

#change the color of the VaR line, as we have red 'x' for the Braech
fig_6_d.update_traces(selector=dict(name='Estimate σ with Lambda=0.99, 1D'), line=dict(color='or
fig_6_d.update_traces(selector=dict(name='Estimate σ with Lambda=0.72, 1D'), line=dict(color='gr

#show chart of Return, VaR and mark 'x' on Breaches
fig_6_d.show()
```



$$\sigma^2_{t+1|t} = \lambda\sigma^2_{t|t-1} + (1-\lambda)r^2_t$$

"EMWA" which stands for Exponentially Weighted Moving Average is used for calculating volatility in time series data. As per the image above, we can see the impact of the λ is shown in the volatiliy of the estimated σ (shown in green line vs orange line), which points that λ determines how much weight is given to observations when calculating volatility estimate.

The more higher the λ is, the more smooth the estimated σ calculated, and the as λ goes to a lower value, it will provide more volatile estimated σ. Based on the mathematical formula above, if λ is very high, the projected result will be weighted more in the prediction of variance ($\lambda\sigma^2_{t|t-1}$ side of the formula), rather than the weights in the previous days return ($(1-\lambda)r^2_t$ side of the formula). Therefore, if λ is high, EMWA-predicted volatility will be less sensitive to recent return data/market movement, which makes smoother estimates, resulting changes in σ to happen gradually over time (vice versa).

END OF EXAM